

# marola-bench: a proposal for a WeatherBench for Brazil

One scorer, Brazilian truth, and a ranking rule written before the scores

Matheus Hoffmann

2026-10-14



OUROBORIC.DEV

## Abstract

Public weather benchmarks made forecast models comparable, but for a person on a Brazilian beach three things are missing in them: a region that is Brazil, a score for ocean waves, and an instrument as the truth. This paper proposes marola-bench, which keeps the method of WeatherBench, WeatherBench 2 and Operational WeatherBench and changes those three things. The bench is defined as the set of score series: the scores of any forecaster against the observations of a point, in order of time. Observation points are added by hand, and one actor for each point asks the forecasters and registers the scores. The paper states seven hard rules of the owner: five for the implementation (Apache Pekko and Scala 3, GCP, Besom, a home in this repository, a cheap and reliable cluster), one for its terms (all rights reserved) and one for its coding assistant (Ponytail 5, a Claude Code plugin). It says where they change the accepted design, MIP-0091. It reports no result: no forecast has been scored, and nothing has been built.

**Cite this as.** Hoffmann, M. (2026). *marola-bench: a proposal for a WeatherBench for Brazil*. Zenodo. <https://doi.org/10.5281/zenodo.23352633>. All rights reserved: read it and cite it; do not copy it or share it without the author's acknowledgement.

**What this is.** A proposal, for a reader outside marola. It says what the benchmark is for, what it copies, and what it adds. It reports no result: no wave forecast has been scored.

**Where the design is.** The design is marola's MIP-0091, accepted by the owner on 13 October 2026 with the note "needs refinement" (Hoffmann 2026b). This paper is the short public account of it. If the two disagree, the MIP is right and this paper is out of date.

**The logos.** The logos of marola and of Ouroboric on the first page are trademarks of their projects. They say where this work is from and which way of working it follows. They do not say that either project endorses this text.

**How to read the marks.** (v) means that the claim was checked against the text of its source while this paper was written. REFERENCES.md gives the passage and says how much of the source was read: the full text, or its abstract. (MIP §n) means that the claim is the design's, from that section of MIP-0091; the MIP keeps its own list of what it checked and what it did not. Δ means not checked by anyone yet.

## Hard rules

These seven are the owner’s rules (Hoffmann, 13 October 2026; rule 7, 14 October 2026). They are not proposals, and they are not open to an agent’s choice. Work on marola-bench that breaks one is wrong, even when it works.

1. **marola-bench is implemented in Apache Pekko, in Scala 3.**
2. **marola-bench is implemented on GCP (Google Cloud), and developed with Claude Code, with an SDD (spec-driven development) framework.**
3. **Its infrastructure as code is defined with Besom.**
4. **marola-bench lives in marola-research/marola-bench.** The first rendered paper is there.
5. **The deployment of the Pekko cluster is cheap, and it is reliable.** Both are hard requirements, and so is its infrastructure on GCP.
6. **marola-bench is all rights reserved, and its credit is the author’s.** It is not shared outside the organisations marola-dev and ouroboric-dev without his acknowledgement. Whoever uses it, a person or an agent, cites it.
7. **Ponytail 5 is adopted as the coding assistant, a Claude Code plugin.**

What each rule meets today, read on 13 October 2026 (rule 7 on 14 October). Five of the seven change the accepted design, MIP-0091. By marola’s own rule, a change like that is written in a MIP first. Until MIP-0091 is amended or a new MIP is accepted, this paper states the rules and the MIP states the older design. The rules are the owner’s later word, so the rules win; the list below is what the next MIP must settle.

### Rule 1: Pekko, Scala 3.

- Scala 3 is already the language: marola-app’s build sets Scala 3.9.0 (`build.sbt` at commit `f769245`) (**v**).
- Pekko is new. That build uses Kyo 1.0.0-RC7 and has no Pekko dependency (**v**), and MIP-0091 describes the forecaster on “the same toolchain (Scala 3, Kyo RC7, JDK 25)” (**MIP §4.4**).
- Apache Pekko is a toolkit for “highly concurrent, distributed, and resilient message-driven applications using Java/Scala”, under Apache-2.0 ([The Apache Software Foundation, n.d.a](#)) (**v**). Its current release is 1.7.1, and its documentation says: “Pekko has support for Scala 3. Pekko is built with Scala 3.3 LTS version” (**v**). So the two halves of the rule fit each other.
- Not said by the rule, and not guessed here: which parts of Pekko (actors, streams, cluster), and how Pekko and the Kyo module live together.

### Rule 2: GCP, Claude Code, an SDD framework.

- GCP is new, and it meets the design head-on. MIP-0091 says “Zero cloud spend” in its header, and its requirement R17 says “Zero spend by construction ... Nothing provisions”: fetched forecasts are free, scoring runs on GitHub Actions, computed runs use owned hardware, and the record is a lake on B2 (**MIP header, R17, §5.2**). With rule 2, something is provisioned and something is paid. What runs on GCP, what stays where it is, and the budget are for the MIP.
- Claude Code is already how the design was written: MIP-0091’s header says “Written by: Hoffmann, with Claude Code” (**MIP header**).
- SDD is new as a rule. MIP-0091’s header has a field “Spec-kit” with the value “none” (**MIP**

**header**). The rule does not name the framework. GitHub’s Spec Kit is one, “to help you get started with SDD”, under MIT ([GitHub, n.d.](#)) (**v**); that the field’s name refers to it is  $\triangle$ . Which framework is the owner’s to name.

### **Rule 3: Besom.**

- Besom is “a Pulumi SDK for Scala”, by VirtusLab, under Apache-2.0 ([VirtusLab, n.d.](#)) (**v**). Pulumi is infrastructure as code in general-purpose languages ([Pulumi Corporation, n.d.](#)) (**v**). So the infrastructure is written in the same language as the benchmark.
- Its own page says that Scala support is “currently in Public Beta”, and its latest release is v0.5.2 of 18 September 2026 (**v**). A beta under a rule is a risk to name, not a reason to break the rule.
- MIP-0091 defines no infrastructure as code, because it provisions nothing (**MIP R17**).

### **Rule 4: marola-research/marola-bench.**

- MIP-0091 decided “zero new repositories”, with two modules in marola-app, and gave marola-research the public copy, the replay results and the reports’ sources (**MIP §4.4**).
- The rule adds no repository, which agrees with the MIP. It moves the benchmark’s home from marola-app to this repository, which does not. Whether the code moves, or only the research and its papers, is for the MIP. Today this directory holds the paper and nothing else.

### **Rule 5: cheap and reliable.**

- The design already has both words, in another form: “Zero spend by construction” and a fallback that is “the normal state” (**MIP R17, §8**). Rule 5 keeps the two aims and changes the means: a cluster that runs all the time, on a cloud that is paid.
- No price has been computed, so “cheap” has no number yet  $\triangle$ . Section 4.4 says how the design makes the cluster cheap and reliable; the number is fixed before anything is deployed.

### **Rule 6: all rights reserved.**

- The accepted design asks for a public chain: scorecards under CC BY 4.0 and a dated data set for each version of the protocol (**MIP R20**). Section 7 of this paper argues the same from the author’s own book.
- Rule 6 is about something else and something that exists: this paper, its notes, its journal and its website. They can be read and cited; they cannot be copied or shared without the author’s acknowledgement. The rule says nothing yet about scores, because there are none.
- The two meet when the first scorecard exists. Whether it is published under CC BY 4.0, as the design says, or under these terms, is for the MIP and for the owner.
- What rule 6 does not undo: this text was under CC BY 4.0 until 13 October 2026, and a copy taken then keeps that licence.

### **Rule 7: Ponytail 5.**

- Ponytail is a plugin for Claude Code by Dietrich Gebert, under MIT ([Gebert, n.d.](#)) (**v**). Its skill calls itself “Lazy senior dev mode: the smallest change that fully solves the task” (**v**). Its release of 14 October 2026 here is v5.2.0, at commit 2961b52 (**v**).
- This repository enables it for every session: `.claude/settings.json` sets `ponytail@ponytail` to `true` (**v**). That file names no version, so “5” is kept by the rule, not by the file. Whether a version 6 is taken is the owner’s decision.

- It agrees with rule 2, which already says “developed with Claude Code”. It agrees with MIP-0091, which says “Written by: Hoffmann, with Claude Code” and names no plugin (**MIP header**).
- It does not settle rule 2’s SDD framework. Its skill file is about the size of a change: it takes the first option that works, from “does it need to exist?” to “the minimum code that works” (**v**). The word “spec” is not in it (**v**). This is the writer’s reading: Ponytail decides how small a change is, and an SDD framework decides what the change is for.
- Its page reports “-53% code, -41% time, -26% cost, -45% tokens”, measured by its author on 39 tasks in Claude Code (**v**). These are its author’s numbers. Nobody has measured them on marola-bench, which has no code yet  $\triangle$ .

## 1. Summary

Public benchmarks made weather forecast models comparable. WeatherBench is “a benchmark dataset for data-driven medium-range weather forecasting”, with evaluation metrics and baseline scores ([Rasp et al. 2020](#)) (**v**). WeatherBench 2 adds an open evaluation framework, ground truth and baseline data, and a website that is updated with the latest metrics ([Rasp et al. 2024](#)) (**v**). Operational WeatherBench scores operational models as they run ([Brightband 2026](#)) (**v**).

For a person on a Brazilian beach, three things are missing in them (**MIP §4.1**):

1. **No region is Brazil.** The selector of Operational WeatherBench “offers the global mean plus the tropics, the northern and southern hemispheres, North America, and Europe” (**v**). The page says that scores are computed for more regions than it shows; neither “South America” nor “Brazil” is on it (**v**).
2. **No score is for waves.** The methodology page names only variables of the atmosphere: geopotential, temperature, wind and precipitation (**v**). marola forecasts sea conditions ([Hoffmann et al. 2026](#)).
3. **No truth is an instrument.** Each model is scored against its own analysis (**v**). Over an ocean with few observations, that favours agreement with the analysis, not with the sea (**MIP §4.2**).

marola-bench is the proposal to keep the method of those benchmarks and change those three things: Brazilian regions, wave variables, and instruments on the Brazilian coast as the truth.

## 2. What is copied, and from whom

The method is not new and this proposal does not claim it. It is the discipline of Operational WeatherBench’s methodology page (**v**):

- **One scorer.** Every model is scored by the same code. Operational WeatherBench uses Google Research’s WeatherBench-X, which is open under Apache-2.0 ([Google Research, n.d.](#)) (**v**).
- **A truth that is stated one time.** There, each model against its own analysis, and precipitation always against ERA5 (**v**). ERA5 is ECMWF’s reanalysis for the Copernicus Climate Change Service, a record of “the global atmosphere, land surface and ocean waves” ([Hersbach et al. 2020](#)) (**v**): it has waves, which matters for section 9.
- **Fixed leads and fixed regions.**
- **A baseline with no skill:** a climatology “is scored as a reference ‘no-skill’ curve” (**v**).

- **A ranking rule that is written down.**

It also copies the licence: the dashboard and its scores are CC BY 4.0, and third-party data keep their own terms (v).

### 3. The proposal

Each item below is a requirement of MIP-0091 (its section 4.3, R1 to R20). The numbers and names are the MIP's.

#### 3.1 Truth is an observation at a point

Truth is what an instrument measured: weather stations, wave buoys and radiosondes on and off the Brazilian coast. Each row of truth carries its instrument, its averaging, its quality flag and its licence tier. A missing observation is a row that says “missing”; it is never replaced by a model's value (**MIP R4**).

There is a network of buoys along the Brazilian coast with meteorological and oceanographic sensors. In 2017 a paper counted “ten currently operational buoys” and analysed the waves at three of them, in the southern part ([Pereira et al. 2017](#)) (v). They are few: the design counts at most about nine buoys of the national programme at the same time (**MIP §4.2**). The format of their files, their averaging and their latency are  $\Delta$  (**MIP, “Not checked”**).

#### 3.2 A licence tier on every row

Some truth is under terms that restrict redistribution. Scoring uses every tier; the published export filters by tier; and the scorecard says which cells it cannot show (**MIP §4.2**). A reader can see which scores he can reproduce and which he cannot. The status of any technical cooperation agreement is  $\Delta$  (**MIP, “Not checked”**), and none is claimed here.

#### 3.3 Regions are marola's own cells

South, Southeast and Northeast, and inside them the buoy, the station and the beach. The scorecard is a projection of those cells (**MIP §4.2**). No page compares a score against instruments with a score from Operational WeatherBench: the truths differ (**MIP R19**).

#### 3.4 Waves are first-class

Significant wave height, peak period and direction, with buoys as truth and their own climatology floor. Wind rows and wave rows stay apart (**MIP R5**).

#### 3.5 Scores that the literature already defines

- A fair continuous ranked probability score (CRPS) with ensembles of 16 members (**MIP R2**). The CRPS is a proper scoring rule ([Gneiting and Raftery 2007](#)) (v). For an ensemble, the usual CRPS is “unfair” and an adjusted version is “fair” ([Ferro 2014](#)) (v).
- For waves, a threshold-weighted CRPS above 2.5 m, because the high sea is the one that hurts (**MIP R5**). A weight on the score “emphasizes regions of interest ... while retaining propriety” ([Gneiting and Ranjan 2011](#)) (v).

- Always a climatology floor, and the reward is skill relative to it. A cell with no floor publishes no reward (**MIP R6**).
- For 24-hour precipitation, Operational WeatherBench uses SEEPS (**v**), a score “for monitoring precipitation forecasts” (Rodwell et al. 2010) (**v**).

### 3.6 One fixed, versioned, public protocol

Leads every 6 hours to 360 hours; scorecard days 1, 3, 5, 7, 10 and 15; ranking on the 12Z cycle; windows of 7, 30 and 90 days; a minimum number of cases. A change is a new version of the protocol, never an edit (**MIP R2**). The parameters of the decision (section 5) are committed before the first replay: this is pre-registration (**MIP R3**).

### 3.7 Few instruments, said honestly

With few buoys, most cells have too few cases for months. Such a cell names no winner. It publishes the median of the admissible models and calls it model agreement. The design says: “The fallback is the normal state”, and the page must not make it look better than it is (**MIP §8**).

## 4. The system: the bench, the points and the actors

This section is the owner’s design of 13 October 2026, in his words where they are quoted. It is later than MIP-0091 and is not in it.

### 4.1 What the bench is

“The bench are the score series performed by any Forecaster, Registered by an Actor.”

A **forecaster** is anything that gives a forecast for a point: a provider that publishes a model today, and marola’s own forecaster later. A **score** is one forecast of one forecaster compared with one observation. A **score series** is those scores in order of time, for one forecaster, one observation point, one variable and one lead. **The bench is the set of all the score series.**

“The bench is the final goal.” The points, the actors, the cluster and the cloud exist to make the series, and nothing else is the product.

Two things follow:

- **marola’s forecaster is one forecaster among the others.** The bench scores it by the same rule and registers it the same way. That agrees with the design: the forecaster reads scores and writes nothing that the scorer reads (**MIP R1**).
- **A series is a record, not a table that is computed again.** Once a score is registered it is not edited. A new protocol version starts new series (**MIP R2**).

### 4.2 Observation points are added by hand

An observation point is added, changed and removed by a person, through a CRUD API (create, read, update, delete) that is specified in Apache Pekko and Scala. There is no automatic discovery of points. Pekko has a module for it: Pekko HTTP is “a full server- and client-side HTTP stack on top of pekko-actor and pekko-stream” (The Apache Software Foundation, n.d.g) (**v**).

By hand, because a point is not only a position. It has an instrument, an averaging, a quality flag and a licence tier (**MIP R4**), and a person must read the terms before a point’s data is scored and shown. Today the design keeps the points in a file, **sampling-points.json** (**MIP**, “**Checked live**”); with this rule they become a resource with an API. How the file and the API meet is for the MIP.

### 4.3 Actors

One actor for each observation point. The actor:

1. **monitors its point**: it knows when the next observation is due, and it gets it;
2. **asks the providers** for their forecasts for that point;
3. **in the future, asks marola’s forecaster** in the same way;
4. **registers** each score in its series.

The actor is the unit of the system. A point that is added through the API becomes an actor; a point that is removed stops one. Nothing in the system is “the batch”: each point keeps its own time.

This is a change of shape. In the accepted design, a cycle runs every four hours on GitHub Actions and scores everything (**MIP §2, §5.2**). Here the work is spread over actors that live in a cluster. The scores and their rules do not change; who runs them does.

### 4.4 Cheap and reliable: the design principles

The Reactive Manifesto names four properties of a reactive system ([Bonér et al. 2014](#)) (**v**): it is responsive, resilient, elastic and message driven. Each one is used here for the two hard requirements.

| Property, in the manifesto’s words                                                                  | What it means here                                                                                    | Cheap                                 | Reliable                                                         |
|-----------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|---------------------------------------|------------------------------------------------------------------|
| <b>Message driven:</b><br>“asynchronous message-passing to establish a boundary between components” | an actor for each point; an actor shares no state; a provider is asked by a message with a time limit | an actor that waits uses no processor | one failed point or one failed provider does not stop the others |

| Property, in the manifesto's words                                  | What it means here                                                                                                                                                                                                                                                                                                                                                             | Cheap                                                                                                  | Reliable                                                                                           |
|---------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------|
| <b>Resilient:</b> “stays responsive in the face of failure”         | an actor that fails is restarted when a supervision strategy says so; with none, a typed actor is stopped ( <a href="#">The Apache Software Foundation, n.d.e</a> ) (v). An actor's state is the events it stored: it “is recovered by replaying the stored events”, also when it is “migrated within a cluster” ( <a href="#">The Apache Software Foundation, n.d.d</a> ) (v) | a node may be lost, so capacity that can be taken away is usable                                       | no score is lost with a node                                                                       |
| <b>Elastic:</b> “stays responsive under varying workload”           | an actor with nothing to do is stopped and started again when needed. Pekko calls this passivation: “you may stop entities that are not used to reduce memory consumption”; by default an idle entity is passivated after 2 minutes ( <a href="#">The Apache Software Foundation, n.d.c</a> ) (v)                                                                              | a point acts a few times a day and is idle the rest; memory follows the work, not the number of points | adding points does not need a larger design                                                        |
| <b>Responsive:</b> “responds in a timely manner if at all possible” | back-pressure: a source “will never emit more elements than the received total demand”, which slows the producer down to the speed of the consumer, with a bounded buffer ( <a href="#">The Apache Software Foundation, n.d.b</a> ) (v)                                                                                                                                        | no capacity is kept for a peak                                                                         | an answer that does not come in time is a row that says “missing”, never a guess ( <b>MIP R4</b> ) |

Six rules follow from them. They are proposals for the next MIP, not facts.

1. **Reliability is what makes it cheap.** Cheap capacity on a cloud is the kind that can be taken away. On GCP, “Spot VMs have significant discounts, but Compute Engine might preemptively stop or delete (preempt) Spot VMs to reclaim the capacity at any time” ([Google Cloud, n.d.](#)) (v). A cluster that survives the loss of a node can use it. So the two hard requirements are one piece of work: first survive the loss of any node, then buy the cheap nodes.
2. **State is events.** An actor stores what happened (an observation came, a forecast came, a score was registered) and builds its state from that. A restart loses nothing.
3. **Every write can be repeated.** After a restart an actor may do its last step again. A score registered two times is one score.
4. **The smallest cluster that survives one loss.** Its size is a value of the protocol, like the budget (**MIP R3, R17**), and it is written before the deployment. A cluster needs a rule for the case in which its nodes cannot see each other, so that both sides “take the same decision about which part will keep running”. Pekko’s Split Brain Resolver has five: keep majority (the default), static quorum, keep oldest, down all, and lease ([The Apache Software Foundation, n.d.f](#)) (v).
5. **The budget has a switch.** The design already has a kill switch that returns every cell to the fallback (**MIP R10**). The same idea for cost: above the budget, the cluster does less (fewer providers, longer intervals) before it costs more, and it says so on the bench.
6. **The infrastructure is code, in the same language.** It is defined with Besom (hard rule 3), so the cluster’s size and its budget are reviewed in a pull request like any other value.

What this section does not decide: which GCP products, where the events are stored, how many nodes, and the budget in money. None has been measured or priced  $\triangle$ .

## 5. From a scoreboard to one forecast

A scoreboard ranks. A swimmer needs one number for one beach. So the design adds a forecaster on top of the scores, and keeps it apart from them: the forecaster reads score cells and writes nothing that the scorer reads (**MIP R1**).

The candidates (the design calls them arms) are of two kinds, and each kind has its own algorithm (**MIP R8**):

- **Fetches arms** are forecasts that others publish. All of them are scored every cycle, so the loss of each is always seen. They are experts with full information, and the rule is Hedge: in each round that algorithm receives the loss of every strategy ([Freund and Schapire 1997](#)) (v).
- **Computed arms** are wave runs that cost compute, such as runs of the WW3 GPU port ([Hoffmann 2026d](#)). Only the runs that were made are seen: “no feedback ... is observed for unchosen arms”, which is what makes a problem a bandit ([Li et al. 2010](#)) (v). The rule is Thompson sampling, which draws an estimate for each action from the posterior and takes the action with the largest ([Russo et al. 2018](#)) (v), with a knapsack: a bandit in which the learner is “constrained by one or more supply (or budget) limits” ([Badanidiyuru, Kleinberg, and Slivkins 2018](#)) (v).

The design is exact on one point: never a bandit over arms whose loss is observed in all cases (**MIP R8**). The owner’s first note called the whole thing a contextual bandit (section 10); the accepted design is narrower, and this paper follows the design.

The context of a decision is the region, the lead time, the variable and the state of ENSO, and every feature of it is dated before the forecast was issued (**MIP R9**).

## 6. Conservative, and safe outside the forecaster

- **A model is admitted for correctness, not for skill.** A wave run enters only with its regression tests passed and recorded; for the GPU port that means bit for bit (**MIP R15**).
- **The quoted model changes slowly.** A new model replaces the incumbent only when the 95% interval of a paired block bootstrap is above zero, on two consecutive weekly runs (**MIP R10**). A block bootstrap resamples blocks of observations, and does not treat a time series as independent values ([Künsch 1989](#)) (v).
- **No live exploration.** A swimmer sees only a model that passed its gate. Exploration changes which computed runs are made, never what is shown (**MIP R11**).
- **Safety reads nothing from the forecaster.** The swimming score and the warning for a dangerous sea do not depend on it (**MIP R12**).
- **Disagreement is shown** beside the winner (**MIP R18**).
- **Zero spend by construction**, in the accepted design: fetched forecasts are free, scoring runs on GitHub Actions, and computed runs use owned hardware under a budget that is a value of the protocol (**MIP R17**). Hard rule 2 puts the benchmark on GCP, so this item changes with the next MIP; a budget that is a value of the protocol is the part to keep.

## 7. Why this is open

Read this section as the design’s argument, not as this paper’s licence. The benchmark’s method and, by the accepted design, its scorecards are public. This paper is all rights reserved (hard rule 6): it is open to read and to cite, and not to copy.

*Closed Source:* *The God that Failed* argues that a result nobody can check is a result nobody should trust ([Hoffmann 2026a](#)). A benchmark is that argument at a small size: a scoreboard with a hidden scorer asks for trust. So the design has a public chain (**MIP R20**):

- the scorecards are CC BY 4.0, with each source’s terms kept;
- each version of the protocol gets a dated data set on Zenodo, with a permanent identifier. The software citation principles ask for an identifier that is “globally unique” and that persists “even beyond the lifespan of the software” ([Smith et al. 2016](#)) (v);
- the reports and their sources are in this repository;
- the toolchain is pinned ([Hoffmann 2026c](#)), so a score can be made again.

## 8. The order

The design’s stages (**MIP §5.6**), in order. No date is given, because the design gives none.

1. Wave truth goes into the evaluation that already runs for wind.
2. The forecaster is written and replayed on the wind record.
3. Computed wave runs enter, after their correctness gate.
4. The product shows the winner, after the gate passed.
5. The public copy, the reports and the Zenodo data sets come to this repository, after the owner approves the publication.

These are the stages of the accepted design. The hard rules and section 4 change who runs the work (actors in a cluster on GCP, not a cycle on GitHub Actions), so the next MIP may change the stages too. This paper does not write new ones.

This paper is before stage 5. It publishes no score and no part of the private policy note that the design cites.

## 9. What is not known

From the design’s own lists (**MIP §8, §11, “Not checked”**):

- the format, averaging and latency of the buoy files, and the terms for each station;
- the status of any cooperation agreement, and what each partner can provide;
- which wave models are served for the Brazilian coast by the public source the design names;
- whether a skill learned in an El Niño season stays valid in a neutral year;
- which buoy is the first wave truth. The design’s default is one off Santa Catarina with a continuous record.

From this paper:

- **What the seven hard rules leave open**, for the next MIP: which parts of Pekko, and how it lives with the Kyo module; what runs on GCP, and its budget, against the design’s zero spend; which SDD framework; and whether the code moves to this repository or only the research does.
- **What section 4 leaves open**: the GCP products, the store of the events, the number of nodes, the budget in money, and how the actors and the four-hour cycle of today meet.
- **The name**. The design calls the benchmark “marola-eval” (**MIP §11**). The owner asked for this paper under the name “marola-bench”. Which one the public sees is his decision.
- **The licence of the scorecards**. This text is all rights reserved (hard rule 6). The design wants CC BY 4.0 for what the benchmark publishes (**MIP R20**). Which of the two the first scorecard gets is the owner’s decision.
- **How deep the reading was**. Each “(v)” was checked against a passage of its source, but for six of the journal papers that passage is the abstract, and for one (the buoys) the abstract on a university’s record, because the publisher’s page asked for a login. `REFERENCES.md` says which. Whether that paper names the national buoy programme is not known.
- **An analysis as a second truth**. ERA5 has ocean waves (section 2). Whether it is good enough on this coast to score against where there is no buoy is not known.

## 10. Where the thesis comes from

- MIP-0091, “The Marola forecaster’s architecture”, in `marola-dev/marola` at commit `cf7bd4f` ([Hoffmann 2026b](#)). Author: Hoffmann. Status: Accepted, 13 October 2026.
- Five notes of the owner in `hoffmann/closed-source` at commit `bc1caaf` (<https://github.com/hoffmann/closed-source/tree/bc1caaf/notes>), filed on 12 and 13 October 2026: the method to copy and its gaps (202610122356), the forecaster as a contextual bandit (202610122357), Brazilian truth (202610122358), exploit in operations and explore in simulation (202610122359), and read, evaluate, adopt for forecast models (202610130000).

How this text was checked: by the `ouro-paper` skill of this repository. Some of its steps were

learned from other people’s skills for paper quality, most of them from the Scientific Agent Skills of K-Dense (Kassib et al. 2026) (v); `.claude/skills/ouro-paper/REA.md` credits each one at a commit.

Left out on purpose: the private policy note that MIP-0091 cites, a future version of WAVEWATCH III, and the names of the institutions behind the agreements. None has a public source.

## References

How each entry was checked is in [REFERENCES.md](#); the list’s source is [refs.bib](#).

- Badanidiyuru, Ashwinkumar, Robert Kleinberg, and Aleksandrs Slivkins. 2018. “Bandits with Knapsacks.” *Journal of the ACM* 65 (3): 1–55. <https://doi.org/10.1145/3164539>.
- Bonér, Jonas, Dave Farley, Roland Kuhn, and Martin Thompson. 2014. “The Reactive Manifesto.” <https://www.reactivemanifesto.org/>.
- Brightband. 2026. “Operational WeatherBench: Methodology.” <https://owb.brightband.com/methodology>.
- Ferro, C. A. T. 2014. “Fair Scores for Ensemble Forecasts.” *Quarterly Journal of the Royal Meteorological Society* 140 (683): 1917–23. <https://doi.org/10.1002/qj.2270>.
- Freund, Yoav, and Robert E. Schapire. 1997. “A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting.” *Journal of Computer and System Sciences* 55 (1): 119–39. <https://doi.org/10.1006/jcss.1997.1504>.
- Gebert, Dietrich. n.d. “Ponytail: Lazy Senior Dev Mode.” <https://github.com/DietrichGebert/ponytail/tree/2961b52fc1fe6a5a87c7a959a79512e5c2f3ffe5>.
- GitHub. n.d. “Spec Kit.” <https://github.com/github/spec-kit>.
- Gneiting, Tilmann, and Adrian E. Raftery. 2007. “Strictly Proper Scoring Rules, Prediction, and Estimation.” *Journal of the American Statistical Association* 102 (477): 359–78. <https://doi.org/10.1198/016214506000001437>.
- Gneiting, Tilmann, and Roopesh Ranjan. 2011. “Comparing Density Forecasts Using Threshold- and Quantile-Weighted Scoring Rules.” *Journal of Business & Economic Statistics* 29 (3): 411–22. <https://doi.org/10.1198/jbes.2010.08110>.
- Google Cloud. n.d. “Compute Engine Documentation: Spot VMs.” <https://cloud.google.com/compute/docs/instances/spot>.
- Google Research. n.d. “WeatherBench-X: A Modular Framework for Evaluating Weather Forecasts.” <https://github.com/google-research/weatherbenchX>.
- Hersbach, Hans, Bill Bell, Paul Berrisford, Shoji Hirahara, et al. 2020. “The ERA5 Global Reanalysis.” *Quarterly Journal of the Royal Meteorological Society* 146 (730): 1999–2049. <https://doi.org/10.1002/qj.3803>.
- Hoffmann, Matheus. 2026a. *Closed Source: The God That Failed*. Zenodo. <https://doi.org/10.5281/zenodo.23333656>.
- . 2026b. “MIP-0091: The Marola Forecaster’s Architecture: The Evaluation Goes First, No New Repo, and the Design Requirements from Read, Evaluate, Adopt on Operational WeatherBench.” <https://github.com/marola-dev/marola/blob/cf7bd4f/docs/MIPs/MIP-0091-marola-forecaster-architecture.md>.
- . 2026c. “Picasso: Self-Contained Nix Flakes for Scientific and Agentic Toolchains, and the NixOS Workstation They Run On.” Zenodo. <https://doi.org/10.5281/zenodo.23311208>.
- . 2026d. “WW3 GPU Lab: Agentic Coding and Agentic Research on a Bit-for-Bit GPU Port of WAVEWATCH III.” Zenodo. <https://doi.org/10.5281/zenodo.23221351>.

- Hoffmann, Matheus, Bruno Valério, Rob Kler Soares da Silva Junior, Elisa Oliveira, Leonardo Ramos Almeida, and Pablo Ribeiro. 2026. “Marola: An Open, Non-Profit Platform for Sea Conditions and Bathing-Water Quality at Brazilian Beaches, Built on Public Data.” Zenodo. <https://doi.org/10.5281/zenodo.23224155>.
- Kassis, Timothy, Vinayak Agarwal, Yuhuan He, Darshil Patel, and Aubrey M. Brueckner. 2026. “Scientific Agent Skills: A Library of Procedural Knowledge for Research Agents.” arXiv:2609.00065.
- Künsch, Hans R. 1989. “The Jackknife and the Bootstrap for General Stationary Observations.” *The Annals of Statistics* 17 (3). <https://doi.org/10.1214/aos/1176347265>.
- Li, Lihong, Wei Chu, John Langford, and Robert E. Schapire. 2010. “A Contextual-Bandit Approach to Personalized News Article Recommendation.” In *Proceedings of the 19th International Conference on World Wide Web*, 661–70. <https://doi.org/10.1145/1772690.1772758>.
- Pereira, Henrique P. P., Nelson Violante-Carvalho, Izabel C. M. Nogueira, Alexander Babanin, et al. 2017. “Wave Observations from an Array of Directional Buoys over the Southern Brazilian Coast.” *Ocean Dynamics* 67 (12): 1577–91. <https://doi.org/10.1007/s10236-017-1113-9>.
- Pulumi Corporation. n.d. “Pulumi: Infrastructure as Code in Any Programming Language.” <https://github.com/pulumi/pulumi>.
- Rasp, Stephan, Peter D. Dueben, Sebastian Scher, Jonathan A. Weyn, Soukayna Mouatadid, and Nils Thuerey. 2020. “WeatherBench: A Benchmark Data Set for Data-Driven Weather Forecasting.” *Journal of Advances in Modeling Earth Systems* 12 (11): e2020MS002203. <https://doi.org/10.1029/2020MS002203>.
- Rasp, Stephan, Stephan Hoyer, Alexander Meroze, Ian Langmore, et al. 2024. “WeatherBench 2: A Benchmark for the Next Generation of Data-Driven Global Weather Models.” *Journal of Advances in Modeling Earth Systems* 16 (6): e2023MS004019. <https://doi.org/10.1029/2023MS004019>.
- Rodwell, Mark J., David S. Richardson, Tim D. Hewson, and Thomas Haiden. 2010. “A New Equitable Score Suitable for Verifying Precipitation in Numerical Weather Prediction.” *Quarterly Journal of the Royal Meteorological Society* 136 (650): 1344–63. <https://doi.org/10.1002/qj.656>.
- Russo, Daniel J., Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. 2018. “A Tutorial on Thompson Sampling.” *Foundations and Trends in Machine Learning* 11 (1): 1–96. <https://doi.org/10.1561/22000000070>.
- Smith, Arfon M., Daniel S. Katz, Kyle E. Niemeyer, and FORCE11 Software Citation Working Group. 2016. “Software Citation Principles.” *PeerJ Computer Science* 2: e86. <https://doi.org/10.7717/peerj-cs.86>.
- The Apache Software Foundation. n.d.a. “Apache Pekko.” <https://github.com/apache/pekko>.
- . n.d.b. “Apache Pekko Documentation: Basics and Working with Flows.” <https://pekko.apache.org/docs/pekko/current/stream/stream-flows-and-basics.html>.
- . n.d.c. “Apache Pekko Documentation: Cluster Sharding.” <https://pekko.apache.org/docs/pekko/current/typed/cluster-sharding.html>.
- . n.d.d. “Apache Pekko Documentation: Event Sourcing.” <https://pekko.apache.org/docs/pekko/current/typed/persistence.html>.
- . n.d.e. “Apache Pekko Documentation: Fault Tolerance.” <https://pekko.apache.org/docs/pekko/current/typed/fault-tolerance.html>.
- . n.d.f. “Apache Pekko Documentation: Split Brain Resolver.” <https://pekko.apache.org/docs/pekko/current/split-brain-resolver.html>.
- . n.d.g. “Apache Pekko HTTP Documentation: Introduction.” <https://pekko.apache.org>

[/docs/pekko-http/current/introduction.html](#).  
VirtusLab. n.d. “Besom: A Pulumi SDK for Scala.” <https://github.com/VirtusLab/besom>.